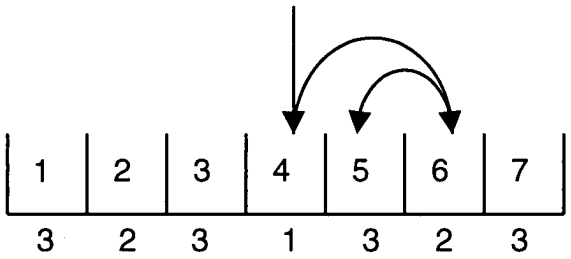


Fig. 1

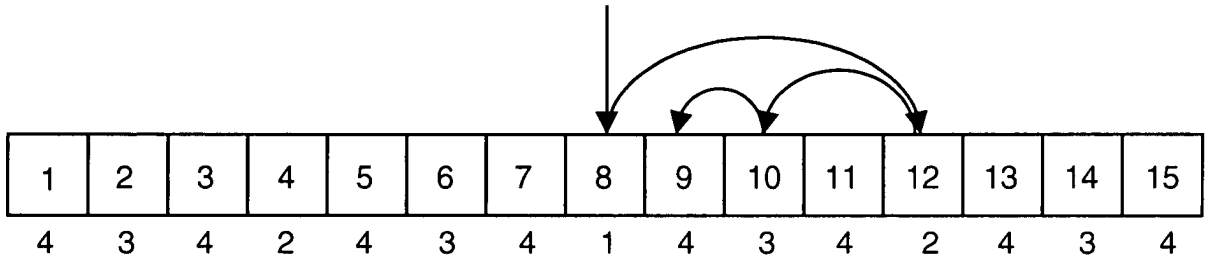


Fig. 2 Unparallelized

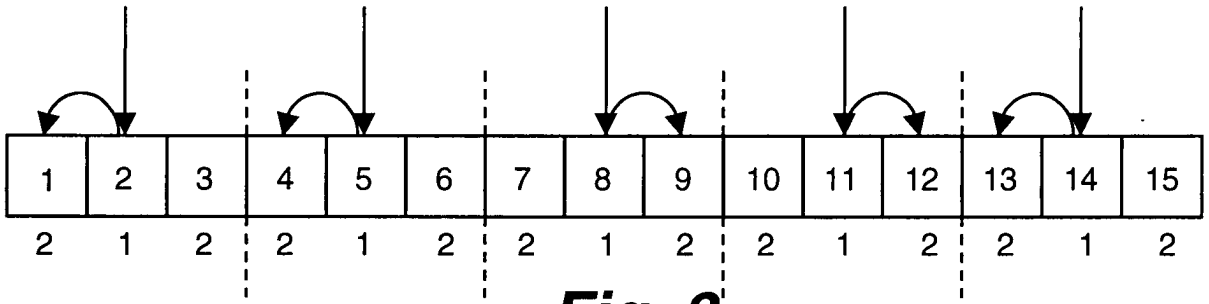
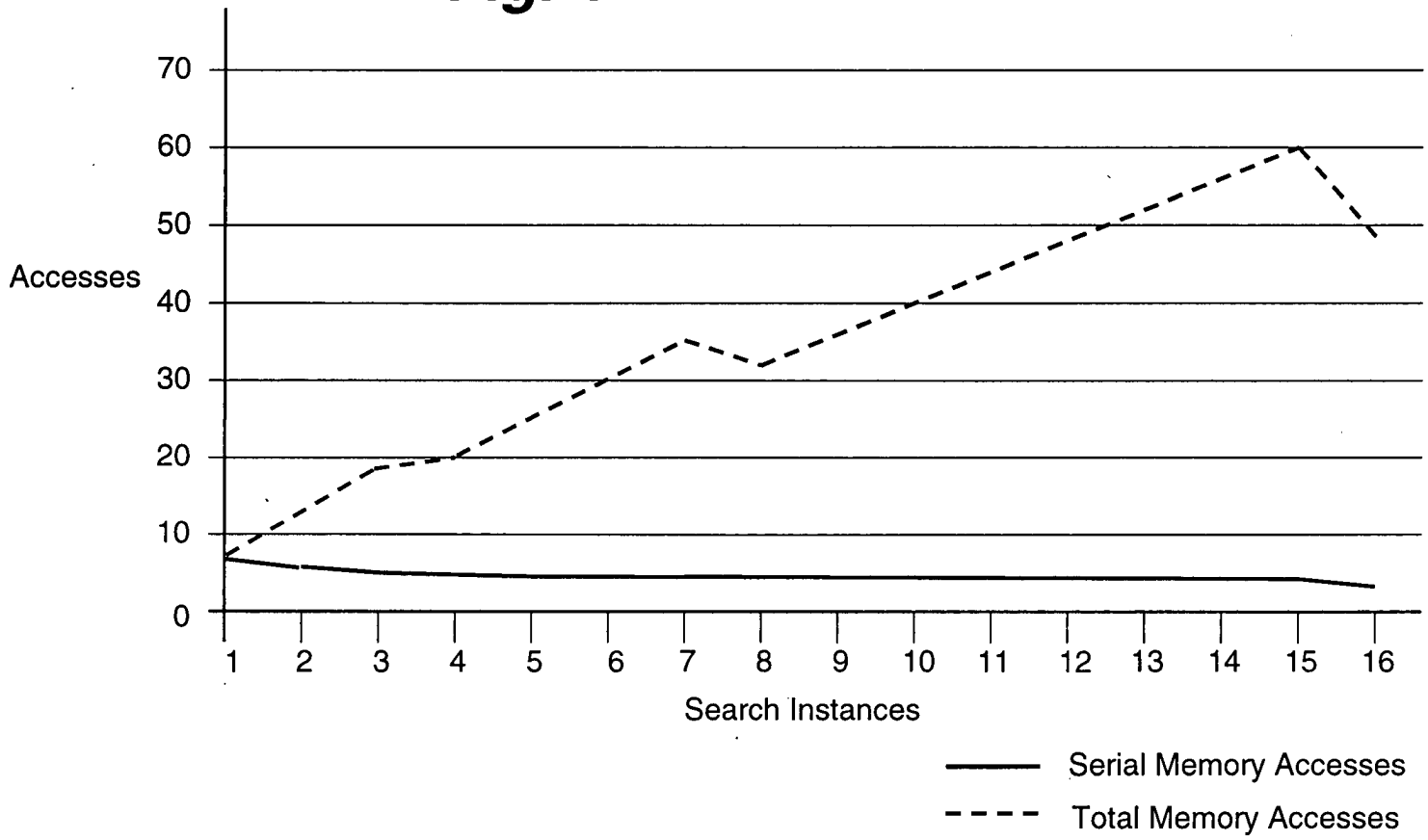


Fig. 3 Parallelized

Fig. 4 Serial and Total Memory Accesses



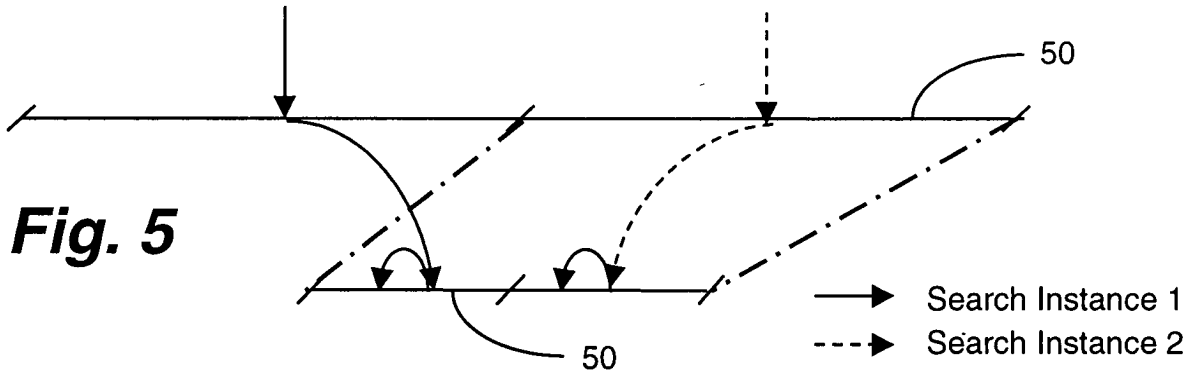


Fig. 5

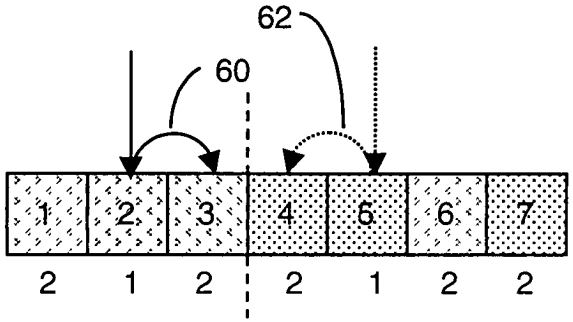


Fig. 6

Search Instance 1
Search Instance 2

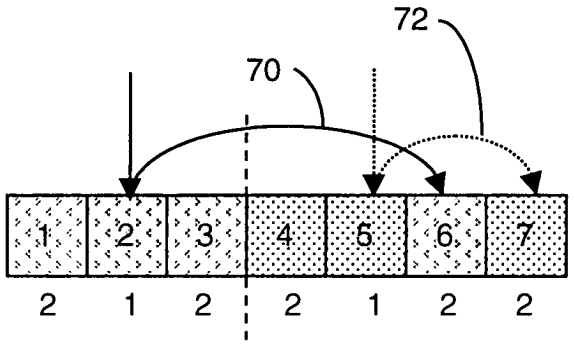


Fig. 7

Search Instance 1
Search Instance 2

Worst Case Lookup	Range 1	Range 2
1	1	1
2	212	2122
3	3231323	323313233233
4	434243414342434	434424344344143424344344244344

Worst Case Lookup	Range 3
1	1
2	21222
3	323331323323332333
4	434442434434443444143424344344243443444344424344344443444

Worst Case Lookup	Range 4
1	1
2	212222
3	32333313233233323333233333
4	434444243443444344443444414342434434424344344434442434434443444434442434434443444434444

Fig. 8

Searchable bins by number of ranges and worst case lookup

Range Sizes

Worst Case Lookup	Range 1	Range 2	Range 3	Range 4
1	1	1	1	1
2	3	4	5	6
3	7	12	18	25
4	15	32	56	88
5	31	80	160	280
6	63	192	432	832
7	127	193	433	833

Fig. 9

Initial range sizes by worst case lookup

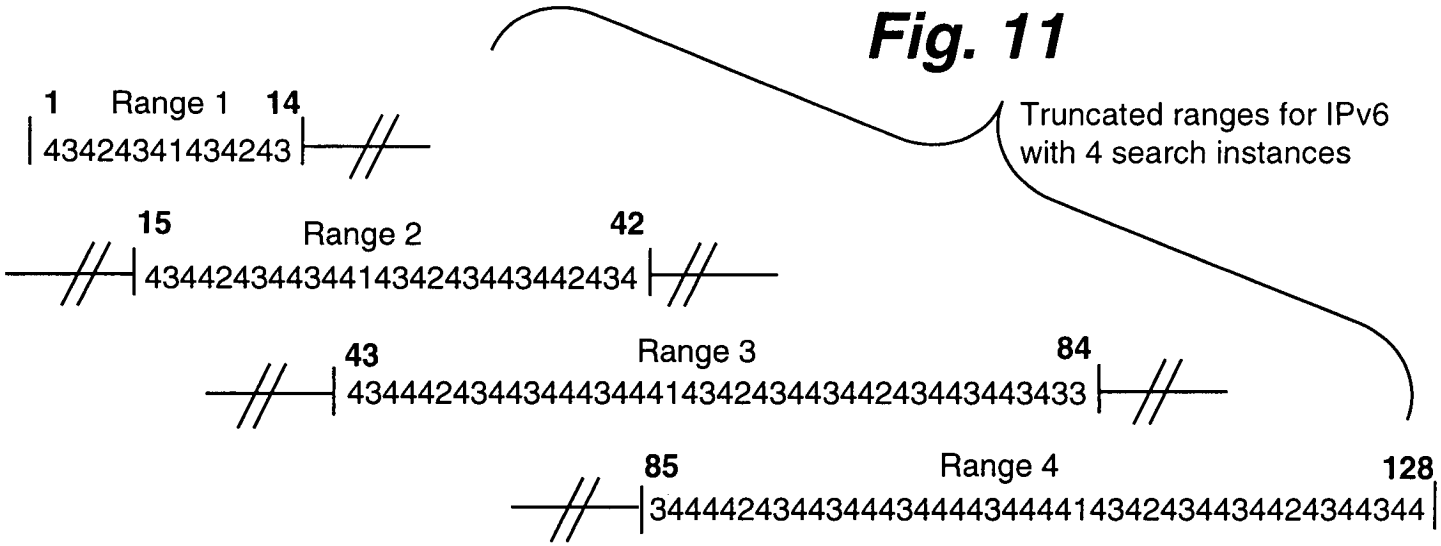
Number of Search Bins

Worst Case Lookup	1 Search Instance	2 Search Instances	3 Search Instances	4 Search Instances
1	1	2	3	4
2	3	7	12	18
3	7	19	37	62
4	15	47	103	191
5	31	111	271	551
6	63	255	687	1519
7	127	320	753	1586

Fig. 10

Number of search bins by worst case lookup

Fig. 11


1
Range 1
14
43424341434243
15
Range 2
42
434424344344143424344244344243443442434
43
Range 3
84
434442434434443444143424344344243443443433
85
Range 4
128
344442434434443444434444143424344344243443443
Truncated ranges for IPv6
with 4 search instances

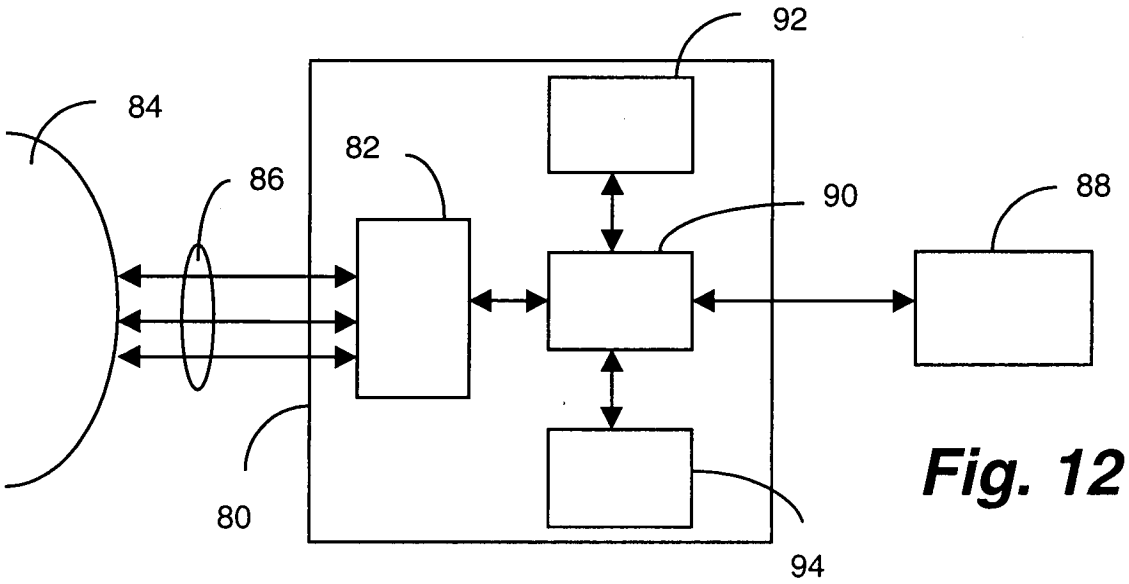


Fig. 12

15 13	12 8	7 5	4 0
RFU	Miss	RFU	Hit

Fig. 13

15 12	11 8	7 4	3 0
RFU	Steal 3	Steal 2	Steal 1

Fig. 14

Pseudo-Code:

```
searchNumber = 0
for each search instance
    initialize(currentInstance.searchValue)

loop {
    searchNumber++
    for each search instance {
        // Perform hash on prefix and initiate hash lookups
        hashLookup(currentInstance.seachValue)
    }

    /* Check the search results of each search instance from highest to lowest. This way if there is a hit at a
    long prefix, the lower instances do not need to be checked. */
    for (numInstances downto 1) {
        // Check if the search was a match
        if (search hit or marker) {
            // Check "Best" bit. If there is no better matches then the search stops here
            if set(Best)
                return (current)
            // If we hit a marker or there are better matches, update state and steal remaining instances
            for ((currentInstance-1) downto 1) {
                stealInstace.searchValue = currentInstance.searchValue + stealTable[stealInstance][searchNumber]
            }
            currentInstance.searchValue += stateTable[searchValue].Hit
            // Break out of the for loop as we no longer care about the rest of the results
            break;
        } else
            // We missed so search shorter prefixes
            currentInstance.searchValue -= stateTable[searchValue].Miss
    }
}
```

Fig. 15